Minttu Wikström

INTEGRAATIOALUSTAN VALVONTA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan insinööri
Joulukuu 2019

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Joulukuu 2019	**Tekijä/tekijät** Minttu Wikström
Koulutusohjelma Tieto- ja viestintätekniikka		
Työn nimi Integraatioalustan valvonta		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 26
Työelämäohjaaja Juho Muuraiskangas		

Tämä opinnäytetyö käsitteli integraatioalustan valvontatyökalu -projektin suunnittelua ja ensimmäisen version luontia, jolla todistetaan ratkaisun käyttökelpoisuus. Tavoitteena oli saada ohjelma, joka pystyy vastaanottamaan tietoja integraatioista ja lähettämään niitä, kun todennuksella vahvistettu käyttäjä pyytää niitä.

Opinnäytetyön alussa käydään läpi valikoituneet työkalut ja niihin liittyvää tietoa. Käymme läpi myös jonkin verran joidenkin työkalujen historiaa. Muutamista tekniikoista on saatavilla myös esimerkki helpottamaan ymmärtämistä. Tämän alustuksen avulla lukija ymmärtää paremmin, mistä tämän projektin toteutuksessa on kyse.

Käytännön osuudessa tutustutaan paremmin projektin suunnitteluun ja toteutukseen. Ensin käydään läpi suunnitteluvaihetta, sen jälkeen toteutusta ja lopuksi testausta ja jatkokehityksen tukitoimenpiteitä. Kaikki osuudet on jaoteltu selkeyden vuoksi omiin alaotsikoihin, jotta lukija saisi mahdollisimman hyvän kuvan projektista

Opinnäytetyö onnistui hyvin haasteista huolimatta. Projekti saatiin vietyä suunniteltuun pisteeseen ja sillä saavutettiin haluttu lopputulos.

Asiasanat
Ansible, Docker, Elasticsearch, Express.js, Keycloak, Node.js, npm, ohjelmointi, VirtualBox.

ABSTRACT

Centria University of Applied Sciences	**Date** December 2019	**Author** Minttu Wikström
Degree programme Information Technology		
Name of thesis SUPERVISION OF THE INTEGRATION BASE		
Instructor Sakari Männistö		**Pages** 26
Supervisor Juho Muuraiskangas		

This thesis describes the designing and proofing concept of the first version of the integration base tracking tool project. The objective was to create program that can receive data of integrations and send the data to authenticated users when they ask for them.

At the beginning of thesis, the information about selected tools will be discussed. Also, there is some history of some tools. Some tools have also examples that will make understanding easier. This information will give the reader a better understanding of what this project is all about.

The practical part will give the reader a better insight into project planning and implementation. First, the design phase, then implementation, and finally the testing and development measures for further development will be discussed. For the sake of clarity, all sections are divided into subheadings to give the reader the best possible picture of the project.

Despite the challenges, the thesis was a success. The project was brought to the planned point and achieved the desired result.

Key words
Ansible, Docker, Elasticsearch, Express.js, Keycloak, Node.js, npm, programming, VirtualBox.

KÄSITTEIDEN MÄÄRITTELY

API	Ohjelmointirajapinta
BUGI	Ohjelman lähdekoodissa oleva virhe
UI	Käyttöliittymä
HTTP	Hypertext Transfer Protocol eli protokolla, jota selaimet ja www-palvelimet käyttävät tiedonsiirtoon
URL	Verkkosivuston tai tiedoston sijainti internetissä
LOGI	Toiminnasta ylläpidettävä tapahtumarekisteri
INSTANSSI	Esiintymä, joka tarkoittaa olio-ohjelmoinnissa luokan edustajaa
KONTTI	Ohjelmapaketti, joka on helppo siirtää ja ajaa
PROVISIOINTI	Automatisoitu ohjelmistojen pohjavalmistelu
SÄIE	Perinteistä prosessia kevyempi prosessi, koska sillä ei ole omia resursseja
INTEGRAATIO	Eri tekniikoilla tai alustoilla toteutettujen ohjelmistojen tai järjestelmien liittäminen yhteen niin, että ne voivat keskustella keskenään

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO 1

2 KÄYTETYT TEKNIIKAT 2
2.1 Node.js 2
2.2 Express.js 3
2.3 Npm 4
2.4 Elasticsearch 6
2.5 VirtualBox 7
2.6 Docker 10
2.7 Ansible 12
2.8 Keycloak 13

3 KÄYTÄNNÖN TOTEUTUS 15
3.1 Suunnittelu 15
3.2 Ohjelmointi 16
3.2.1 Valvonta-API 16
3.2.2 Raportti-API 18
3.2.3 Backend 19
3.2.4 Gateway 20
3.2.5 Autentikointi 20
3.3 Testaus ja jatkokehityksen tukitoimet 23

4 YHTEENVETO 25
LÄHTEET

KUVAT

KUVA 1. Node.js:n tämänhetkinen suunnitelma versioiden julkaisuista, tuen antamisesta ja lopettamisesta 3
KUVA 2. Yksinkertainen esimerkki express.js-koodista 4
KUVA 3. Pakettivaraston käyttöliittymä 6
KUVA 4. Esimerkkikuva virtuaalisesta tietokoneesta käynnissä Red Hat x64 -käyttöjärjestelmällä 8
KUVA 5. Näkymä VirtualBox-ohjelmasta, jossa on auki uuden virtuaalisen koneen luonti-ikkuna 9
KUVA 6. Dockerin kontin ja virtuaalikoneen rakenne-erot 11
KUVA 7. Docker Hubin käyttöliittymä 12
KUVA 8. Esimerkki Ansiblen Playbookista 13
KUVA 9. Kirjautumisnäkymä, johon on lisätty kaikki mahdolliset kirjautumis- ja rekisteröitymisvaihtoehdot, salasanan palautus ja kielen vaihto 14
KUVA 10. Keycloakin järjestelmänvalvojan konsolin perusnäkymä 15
KUVA 11. Alkuperäisen suunnitelman mukainen järjestelmä 16
KUVA 12. Ote valvonta.yaml-tiedostosta 18
KUVA 13. Ote integraatioiden tietojen hausta 19
KUVA 14. Ote integraation lähetyksestä kantaan 20
KUVA 15. Integraatioiden perustietojen haku gatewayn kautta 21
KUVA 16. Näkymä Valvontaclientin asetusten perusnäkymästä 22
KUVA 17. Näkymä haettaessa gatewayn takaa tietoja 23

KUVA 18. Vastaus selaimelta haettaessa integraatioiden perustietoja 24

1 JOHDANTO

Opinnäytetyön tarkoituksena oli tehdä ensimmäinen versio järjestelmästä, jonka avulla voi seurata useita integraatioita. Tämä ensimmäinen versio järjestelmästä todistaa ratkaisumallin käyttökelpoisuuden ja toimii pohjana myöhemmin toteutettavalle seurantatyökalulle. Työ on tehty Alfame Systems oy:lle, jossa on tehty ja tehdään useille eri asiakkaille integraatioita. Niitä on kertynyt jo satoja. Niiden seuraaminen alkaa käydä hankalaksi ja aikaa vieväksi, kun tarkistuspaikkoja on kertynyt jo paljon ja lisää vain tulee.

Järjestelmän tarkoituksena on koota kaikkien integraatioiden tiedot yhteen, jotta tilanteen tarkistaminen olisi helppoa ja vaivatonta. Samasta näkymästä näkee nopeasti integraation tilan ja liikenteen määrän. Järjestelmä myös tietää kunkin integraation käyttötiheyden ja ilmoittaa, jos jokin integraatio on ollut liian pitkään käyttämättä. Näin vältetään turhaa tarkistelua ja estetään suuremmat ongelmat, jotka voisivat aiheutua integraation hiljaisesta kuolemasta.

Järjestelmä on kokonaisuutena niin suuri monen eri osa-alueensa vuoksi, että siitä riittäisi materiaalia useampaan opinnäytetyöhön. Sen vuoksi projekti toteutetaan kevyenä ratkaisuna, joka todistaa ratkaisumallin käyttökelpoisuuden ja jonka pohjalta tätä projektia jatketaan eteenpäin kohti sen lopullista tavoitetta. Sitä varten on myös tehty asianmukaiset dokumentaatiot.

Aluksi esittelen käyttämäni tekniikat siinä muodossa kuin päädyin niitä käyttämään tämän projektin rakentamisessa. Sen jälkeen kerron käytännön toteutuksesta, testaamisesta ja kohtaamistani haasteista. Lopuksi vielä kokoan yhteen projektin ja sen herättämät ajatukset.

2 KÄYTETYT TEKNIIKAT

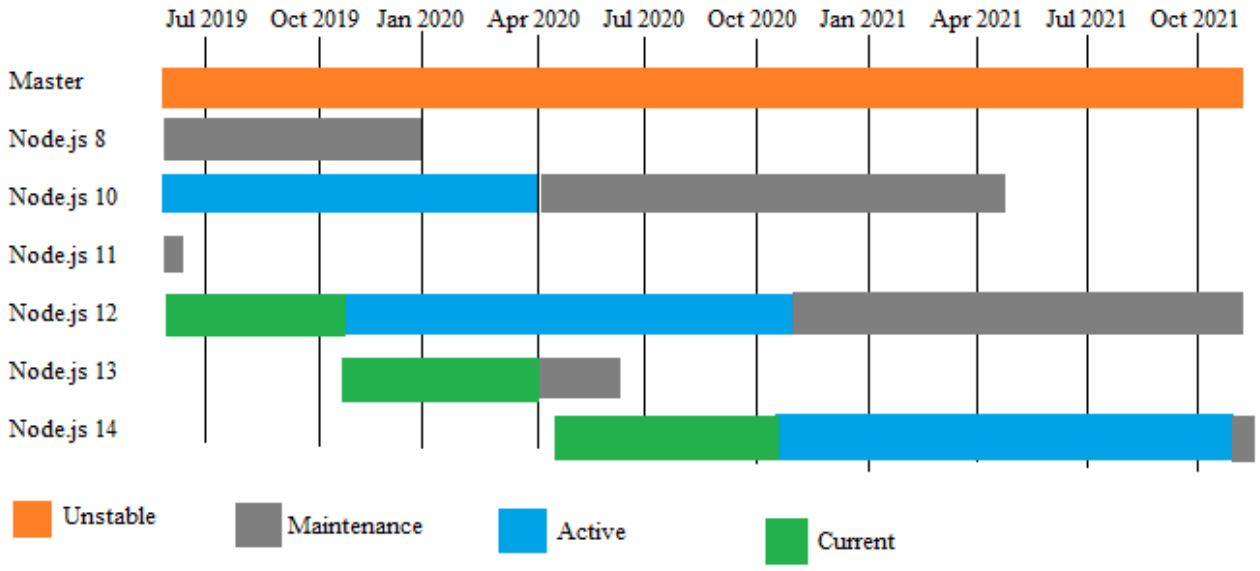
Seuraavaksi tulen käymään läpi tekniikat, joita päädyin käyttämään tämän opinnäytetyön tekemiseen. Suunnitteluvaiheessa tutkin useampia eri tekniikoita löytääkseni parhaimmat tähän projektiin. Lopulta päädyin käyttämään Node.js:ää, Express.js:ää, npm:ää, Elasticsearchia, VirtualBoxia, Dockeria, Ansiblea ja Keycloakia. Käyn kyseiset tekniikat läpi samassa järjestyksessä, kuin ne on tässä mainittu.

2.1 Node.js

Node.js on tapahtumapohjainen ajonaikainen järjestelmä Javascript-ohjemien suorittamiseen. Se sopii hyvin verkkoa hyödyntävien sovellusten ajoon. Se voi vastata useampaan pyyntöön samanaikaisesti mutta siirtyy lepotilaan, kun tehtäviä ei ole jäljellä. Node.js on käytännöllinen niin kehityksessä kuin varsinaisessa käytössä, koska se pystyy käsittelemään useampia pyyntöjä samanaikaisesti. Node.js:ssä I/O-toiminnot tehdään epäsuorasti ajoympäristön ulkopuolella, minkä johdosta vältytään lukitustilanteilta, jotka vaivaavat erityisesti säikeitä käyttäviä sovelluksia. Näin käyttäminen on sujuvaa ja sovellus skaalautuu helposti eri kuormitusasteille. (Node.js 2019a.)

Node.js julkaistiin vuonna 2009. Sen suosiolle on useampia syitä, yksi niistä jo mainittu ylempänä. Muita syitä olivat JavaScript-kielen muuttuminen vakavasti otettavaksi ohjelmointikieleksi ja JavaScript-virtuaalikoneiden suorituskyvyn merkittävä parantuminen. Näiden lisäksi Node.js toi mukanaan innovatiivisen lähestymis- ja ajattelutavan, joka helpotti monen kehittäjän työtä. Node.js on siis kokonaisuudessaan ketterä, tehokas ja helppo kehitykseen. (Node.js 2019a.)

Kuvasta 1 näkee Node.js:n tämänhetkisen tilanteen uusien ja tuettujen versioiden suhteen. Tällä hetkellä virallisesti käytössä ovat versiot 10 ja 12. Molemmat ovat statukseltaan LTS (Long-term-support), joka tarkoittaa sitä, että kriittisille vioille taataan korjaus 30 kuukauden ajan. Ennen active-tilaan siirtymistä versiot ovat current-tilassa, jonka aikana kirjastojen tekijöillä on aikaa lisätä tuki uuteen versioon Node.js:stä. Esitetyt päivämäärät toki voivat vähän muuttua, mutta ne eivät ole lukkoon lyötyjä. Ne kuitenkin antavat kehittäjille hyvää kuvaa siitä, missä suunnilleen mennään versioiden kanssa. Siitä voi hyvinkin nopeasti, kannattaako käyttää jotain versiota tuotannossa, sillä siellä ei kannata käyttää vanhoja versioita, jotka eivät ole enää aktiivisia tai joita ei enää ylläpidetä. (Node.js 2019b.)



Release	Status	Codename	Initial release	Active LTS Start	Maintenance LTS Start	End-of-life
V8	Maintenance LTS	Carbon	30.5.2017	31.10.2017	1.1.2019	31.12.2019
V10	Active LTS	Dubnium	24.4.2018	30.10.2018	1.4.2020	30.4.2021
V12	Active LTS	Erbium	23.4.2019	21.10.2019	21.10.2020	30.4.2022
V13	Current		22.10.2019		1.2.2020	1.6.2020
V14	Pending		21.4.2020	20.10.2020	20.10.2021	30.4.2023

KUVA 1. Node.js:n tämänhetkinen suunnitelma versioiden julkaisuista, tuen antamisesta ja lopettamisesta (Mukaillen Node.js 2019b.)

2.2 Express.js

Express.js-sovelluskehitys julkaistiin vuonna 2010. Se on erittäin suosittu Javascript-pohjaisten web-sovellusten kehittäjien keskuudessa. Sitä kuten muitakin viitekehyksiä käytetään erilaisten http-kutsujen käsittelyssä. Express.js:n avulla pystyy käsittelemään erilaisia http-kutsuja (esimerkiksi POST, GET, DELETE) ja niiden lisäksi URL-osoitteita. Sillä voi myös määrittää yleisiä asetuksia ja malleja, kuten osoittaa mallivastauksen sijainnin, jota hyödynnetään kutsuun vastattaessa. Express.js tukee myös väliohjelmistoa, jota käytetään usein rajapintana. Express.js:lle on tehty myös paljon paketteja ja kirjastoja,

joiden avulla pystytään vastaamaan lähes kaikkiin ongelmiin, joita tulee vastaan web-kehityksessä. Reititysten käyttäminen on myös yleistä. (MDN 2019.)

Express.js:llä tehty http-kutsuja kuunteleva koodi on tehty yksinkertaisimmillaan alla olevan esimerkin (KUVA 2) tavoin. Alussa kerrotaan, että käytetään express.js:ää. Se pitää ensin asentaa npm:n avulla, jotta ohjelma toimii. Riveillä 4–6 on osoite, jota voi kutsua GET-kutsulla. Osoitteena on yksinkertaisesti ”/”. Kun lähettää kyseiseen osoitteeseen GET-kutsun, se palauttaa vastauksena ”Hello to you behind screen!”. Riveillä 8–10 on asetus, jonka mukaan ohjelma kuuntelee osoitetta localhost 8000 ja sinne tulevia pyyntöjä. Koodista löytyvä console.log kirjoittaa komentokehotteen ikkunaan suluissa olevan tekstin ohjelman käynnistyessä. Nyt voi tehdä kutsun selaimessa osoitteeseen ”localhost:8000/” ja vastaukseksi tulee vasempaan yläkulmaan valkoisella pohjalla teksti ”Hello to you behind screen!”.

```
1   var express = require('express');
2   var app = express();
3
4   app.get('/', function(req, res) {
5     res.send('Hello to you behind screen!');
6   });
7
8   app.listen(8000, function() {
9     console.log('App is found from localhost 8000');
10  });
```

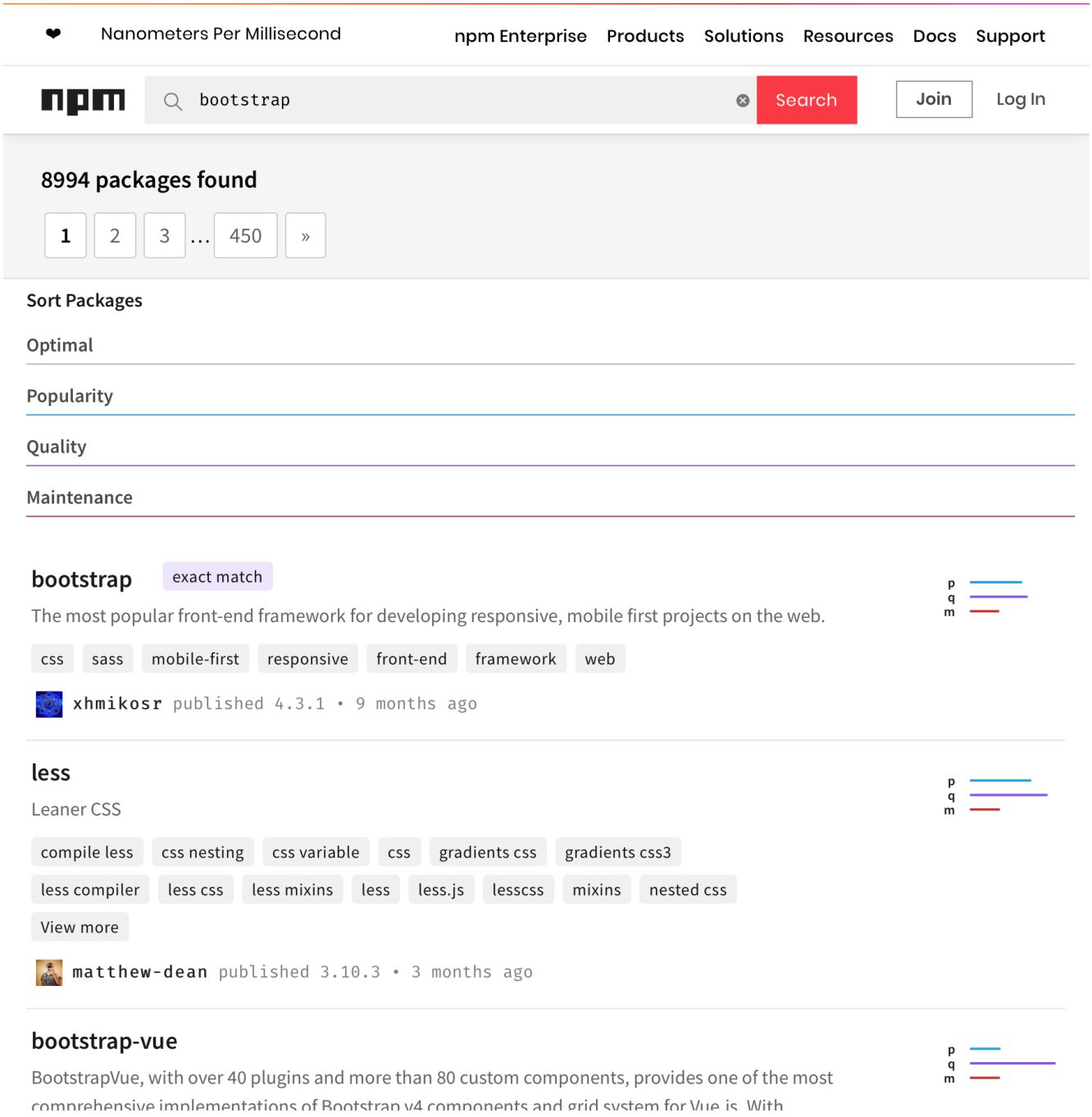
KUVA 2. Yksinkertainen esimerkki express.js-koodista.

2.3 Npm

Npm on suurin ohjelmistorekisteri, jossa on kehittäjien luomia paketteja erilaisiin käyttötarkoituksiin. Käyttäminen on täysin ilmaista; kaikki paketit ovat avointa lähdekoodia, joten niitä voi käyttää vapaasti. Tuki on myös saatavilla npm:n puolesta. Npm tarjoaa myös maksullisen version npm Orgs hintaan 7 dollaria/käyttäjä. Tämä mahdollistaa ilmaisen version mahdollisuuksien lisäksi koodien ja pakettien privaatin jakamisen. Näin voi jakaa paketteja oman tiimin sisällä ja tiimiin pystyy tarvittaessa lisäämään tai vähentämään tekijöitä. Tämän lisäksi npm tarjoaa maksullista jäsenyyttä nimeltään npm Enterprise. Hintaa ei ole kerrottu, mutta sen saa pyytämällä tarjouksen. Lisänä aiempiin tulee tavallista parempi

tuki, alan standardisoitu tunnistautumiseen, laskuperusteinen laskutus ja yksityinen rekisteri. (Npm 2019a.)

Npm koostuu kahdesta osasta: paketinhallintaohjelmistosta ja ohjelmapakettivarastosta. Ohjelmapakettivarasto on julkinen tietokanta javascript-ohjelmista ja niiden metatiedoista. Ohjelmapakettivarastoon pääsee käsiksi käyttöliittymän (KUVA 3) kautta. Siellä voi luoda profiileja, muokata asetuksia ja hakea lukuisia erilaisia paketteja erilaisiin käyttötarkoituksiin. Paketinhallintaohjelmisto puolestaan toimii varsinaisen kehittämisen puolella. Sitä käytetään yleensä komentokehotteesta, ja sen avulla suurin osa lataa, asentaa, päivittää ja suorittaa suoraan projektin sisällä erilaisia paketteja. Paketinhallintaohjelmiston kautta voi suorittaa myös monia muita paketteihin liittyviä toimintoja. Npm:n kautta löytyy kätevästi paketteja erilaisiin käyttötarkoituksiin, ja samaan ongelmaan saattaa helposti olla useampikin eri ratkaisu. Pakettien käyttöönotto paketinhallintaohjelmiston avulla tapahtuu vaivattomasti kuten myös päivitys. Paketinhallintaohjelmiston avulla voi myös jakaa omia paketteja muille käyttäjille. (Npm 2019b.)



KUVA 3. Pakettivaraston käyttöliittymä.

2.4 Elasticsearch

Elasticsearch on vuonna 2010 julkaistu avoimen lähdekoodin haku- ja analyysimoottori. Sitä voi käyttää kaikentyyppisen datan kanssa, kuten jäsennetyn, jäsentämättömän, numeerisen, tekstin ja geospatiaali-

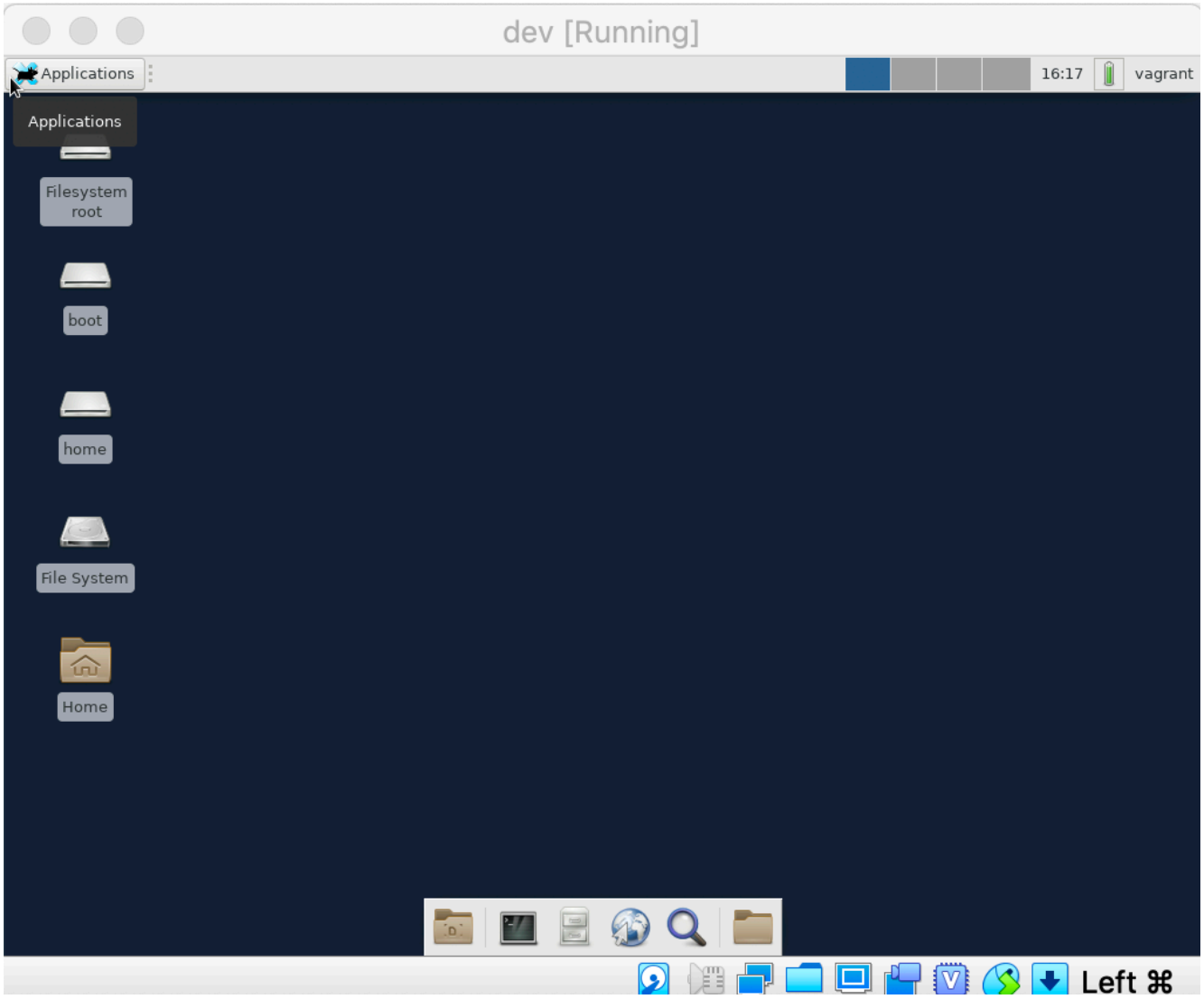
sen datan. Elasticsearchia käytetään esimerkiksi ohjelmien ja nettisivujen hauissa ja erilaisissa analyysien teoissa. Näiden lisäksi on muun muassa erilaisten datojen seurantaa ja visualisointia. Vastaanottaessaan dataa Elasticsearch ottaa vastaan, se ottaa vastaan myös indeksin, jonka perusteella se asettelee datan varastoon. Indeksiä voisi siis verrata tietokantaan. (Elastic 2019a.)

Elasticsearch on monipuolinen työkalu, ja sitä voi käyttää monien eri ohjelmointikielien kanssa sujuvasti. Tuettuja ohjelmointikieliä ovat Go, .NET, PHP, Java, Ruby, JavaScript, Perl ja Python. Tallennettavan datan näkökulmasta Elasticsearchilla on tuki 34 eri kieleen ja lisää on saatavilla lisäosilla. Datan sisältämä mahdollinen teksti ei siis todennäköisesti aiheuta ongelmaa tallennuksen yhteydessä. Näiden lisäksi Elasticsearch on lähes reaaliaikainen ja se myös skaalautuu tehokkaasti. Siinä on todella paljon sisäänrakennettuja ominaisuuksia, jotka tehostavat datan hakemista ja säilömistä. (Elastic 2019a.)

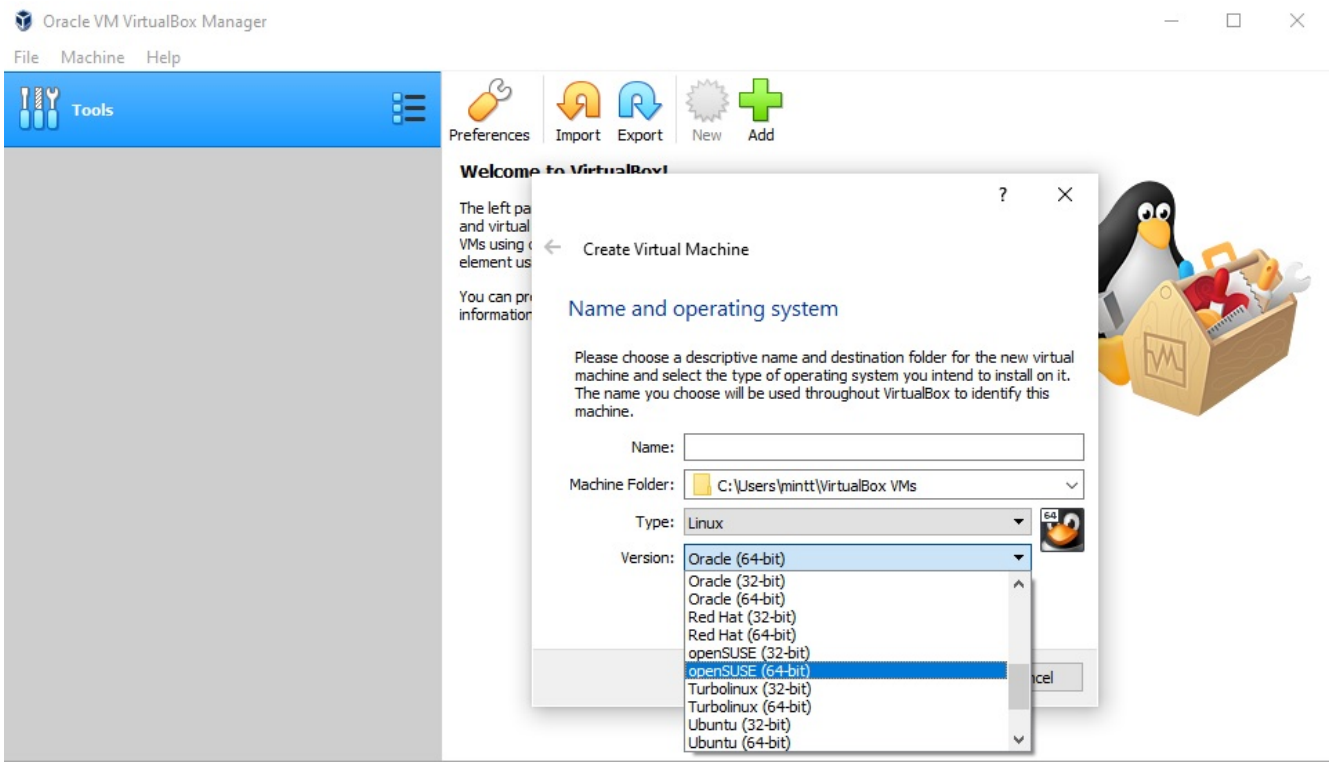
Elasticsearch on ilmainen avoimen lähdekoodin työkalu, mutta jos haluaa saada palveluita ja lisäominaisuuksia, niistä joutuu maksamaan. Elasticsearchin Standard-lisenssi on 16 dollaria kuukaudessa. Sillä saa muun muassa Elasticsearchin ja Kibanan isännöitynä Elasticin puolesta, perussuojauksen ja Elasticin erilaisia ratkaisuita. Seuraavista lisenssitasoista ei ole hintatietoja saatavilla, mutta tasojen nimet ovat Gold, Platinum ja Enterprise. Gold sisältää aiemmin mainittujen lisäksi esimerkiksi toimistoaikojen puitteissa toimivan tuen ja kustomoituja liitännäisiä. Platinumissa on edellä mainitut, paitsi että tuki on koko ajan saatavilla ja lisäksi on koneoppiminen ja kehittyneempiä suojauksia. Enterprisessä on myös kaikki aiemmat ja lisäksi päätepisteen täysi suojaus, tunnistus ja tapahtumien seuranta. (Elastic 2019b.)

2.5 VirtualBox

VirtualBox on ohjelma, jonka avulla voi luoda ja käyttää virtuaalisia tietokoneita haluamallaan käyttöjärjestelmällä. Se tarjoaa tavallaan tietokoneen tietokoneen sisällä. Tarvittaessa niitä voi olla useita käynnissä yhtä aikaa. Virtuaalikoneita voi käyttää näppärästi käyttöliittymän kautta, joka toimii kevyesti kuin mikä tahansa muu ohjelma tietokoneessa (KUVA 4). Käyttöjärjestelmiä on lukuisia jo valmiiksi, ja ne on helppo pystyttää VirtualBoxin ohjelmasta käsin (KUVA 5). Esimerkkeinä voisi mainita Windowsista versiot 95-10, Mac OS X Mojave, Sierra ja High Sierra. Myös Linux-pohjaisia käyttöjärjestelmiä on esimerkiksi Ubuntua, Fedoraa ja Debiania. (VirtualBox 2019.)



KUVA 4. Esimerkkikuva virtuaalisesta tietokoneesta käynnissä Red Hat x64 -käyttöjärjestelmällä.



KUVA 5. Näkymä VirtualBox-ohjelmasta, jossa on auki uuden virtuaalisen koneen luonti-ikkuna.

VirtualBoxin tärkeimpiin ominaisuuksiin kuuluu laaja tuki erilaisille isäntäkoneen käyttöjärjestelmille. Windowsille on tuki käyttöjärjestelmille 7–10, Mac OS X:n tuki alkaa Sierrasta ja Linuxilla on myös useita eri versioita, joilla VirtualBox toimii. Sen lisäksi prosessoreita tai muitakaan tietokoneen osia ei tarvitse virtualisoida ja on mahdollista jakaa kansioita isäntäkoneesta suoraan virtuaalikoneeseen. VirtualBoxista löytyy myös laaja lisäosien tuki, joiden avulla voi virtuaalikoneeseen ottaa käyttöön niin usb-portteja, webkameroita kuin myös levyasemia. Resoluution muuttelu on vaivatonta, koska sitä ei ole sidottu fyysisen näytön mittoihin, vaan sen voi säätää itselle mieluisaan kokoon. (VirtualBox 2019.)

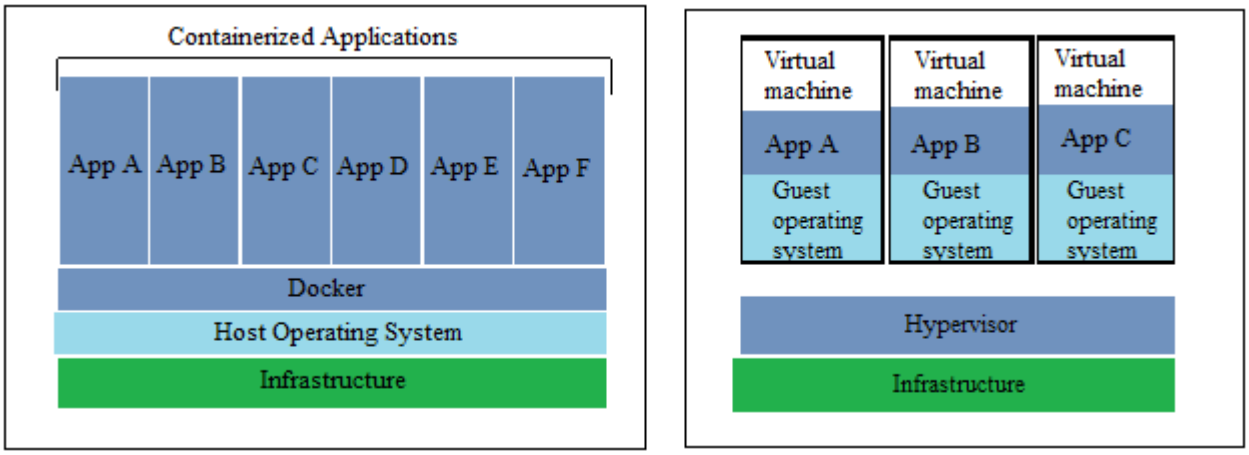
Virtuaalikoneiden käytöstä on monenlaisissa tilanteissa hyötyä. Ohjelmoinnin ja kehityksen näkökulmasta katsottuna nostaisin hyödyistä esille mahdollisuuden testata ohjelmia helposti useilla eri käyttöjärjestelmillä kätevästi yhtä konetta käyttäen. Ei tarvitse asennella eri käyttöjärjestelmiä uudelleen ja uudelleen testaamista varten vaan pystyy käynnistämään haluamansa virtuaalikoneet pystyyn ja testaamaan ohjelmaa nopeasti ja näppärästi sitä kautta. Se nopeuttaa huomattavasti kehittämistä, kun jää turhia työvaiheita välistä ja säästää rahaa, kun ei ole tarvetta useille tietokoneille. (VirtualBox 2019.)

Testaaminen ja täydestä tuhosta selviytyminen on myös helpompaa verrattuna siihen, jos isäntäkoneen kanssa törmättäisiin samoihin haasteisiin. Virtuaalikone on kuin kontti, jota voi vapaasti kopioida, varmuuskopioida, jäädyttää, käynnistää, sammuttaa. Näiden lisäksi virtuaalikoneen hetkellisestä tilasta voi ottaa tarkan tallennuksen, johon voi tarvittaessa palata, jos jokin menee pahasti pieleen. VirtualBox tarjoilee siis hyvin joustavan alustan kaikenlaisia kokeiluita varten ilman, että tarvitsee huolehtia seurauksista. (VirtualBox 2019.)

2.6 Docker

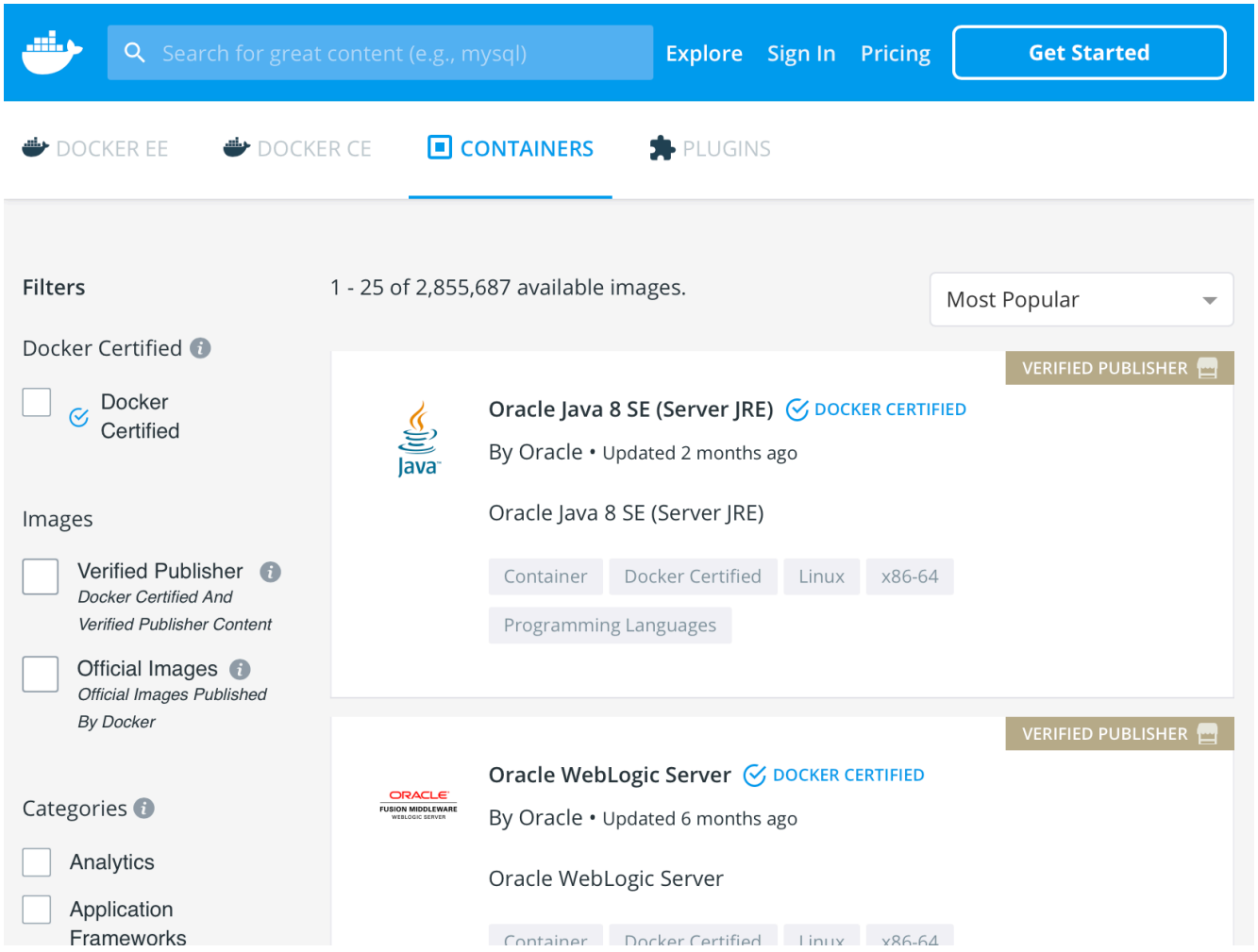
Docker on konttialusta, jonka avulla minkä tahansa ohjelman jakaminen, pystytys ja ajaminen tapahtuu saumattomasti, milloin ja missä tahansa. Kontti sisältää kaiken, mitä ohjelman ajamiseen vaaditaan; ohjelman koodit, järjestelmätyökalut, kirjastot ja asetukset. Dockerin kontin kuva on itsenäinen, kevyt ja ajettavissa oleva sovelluksesta tehty paketti. Docker enginen ajaessa kontin kuvia kuvista tulee kontteja. Dockeria voi käyttää täysin ilmaiseksi, mutta siihen on myös saatavilla maksullinen versio nimeltään Docker Enterprise. Hintaa ei ole julkaistu, mutta sen saa pyytämällä. Tärkeimpänä ominaisuutena enterprise-versio tuo suojauksen jokaiseen ympäristöön paikallisesti hidastamatta mitään toimintoa. (Docker 2019a.)

Kontit ja virtuaalikoneet muistuttavat hyvin paljon toisiaan, kun tarkastelee allokoinnin etuja ja resurssien eristämistä. Eroja alkaa syntyä, kun katsotaan toimintatapoja, sillä kontit virtualisoivat käyttöjärjestelmän, kun taas virtuaalikone virtualisoi laitteiston, kuten edellisessä luvussa huomattiin. Kontit yleisesti ottaen ovat tehokkaampia ja helpommin siirrettäviä. Rakenne kontilla ja virtuaalikoneissa olevista ohjelmista myös eroaa huomattavasti (KUVA 6). (Docker 2019a.)



KUVA 6. Dockerin kontin ja virtuaalikoneen rakenne-erot. (Mukaillen Docker 2019a.)

Docker tarjoaa myös palvelun nimeltään Docker Hub. Siellä on yhteisön, ohjelmistotoimittajien ja avoimen lähdekoodin projektien konttikuvia yhteensä yli 100 000. Osa niistä on Dockerin sertifioimia, ja ne saavat luotetun ja tuetun leiman (KUVA 7). Docker Hubista saa ilmaiseksi ottaa käyttöön konttikuvia, mutta omien konttikuvien säilöminen, pystytys ja jakaminen on maksullista. Docker Hubilla on hinnasto niin yksittäiselle ihmiselle kuin organisaatioille. Yksityiselle hinnat ovat ilmaisen ja 500 dollarin väliltä. Ilmaisella saa yhden privaatin säilytyspaikan ja yhden rinnakkaisen pystytyksen. 7 dollarin hintaan saa 5 privaattia säilytyspaikkaa ja samanaikaisesti voi pystyttää 5 konttikuvaa. Kaiken kaikkiaan hintaluokkia on 8, ja kalleimman 500 dollarin sopimuksella saa 500 privaattia säilytyspaikkaa ja voi yhdenaikaisesti pystyttää kaikki 500 konttikuvaa. Yrityksille on muuten sama hinnasto, mutta sieltä puuttuu ilmainen versio. Hinnasto siellä on siis 7 dollarin ja 500 dollarin välillä. (Docker 2019b.)



KUVA 7. Docker Hubin käyttöliittymä.

2.7 Ansible

Ansible tuo automaation projekteihin. Se säästää aikaa ja nopeuttaa työskentelyä, kun usein toistuvaan ongelmaan on jo tehty ratkaisu ja sen voi helposti ajaa tarvittaessa uudestaan ja uudestaan. Manuaalisen työskentelyn poistuttua virheiden määrä laskee, työn mielekkyys kasvaa ja aikaa jää enemmän innovaatioihin. Ansible on voimakas ja yksinkertainen työkalu. Luominen on yksinkertaista, koska koodaamista ei tarvitse osata ja teksti on täysin ihmisen luettavissa. Selkeyttä tuo tehtävien suoritus siinä järjestyksessä kuin ne on kirjoitettu ansible-tiedostoon. Ansiblea käytetään esimerkiksi ohjelmien käyttöönotossa, provisioinnissa, jatkuvassa julkaisussa ja turvallisuudessa. (Ansible 2019a.)

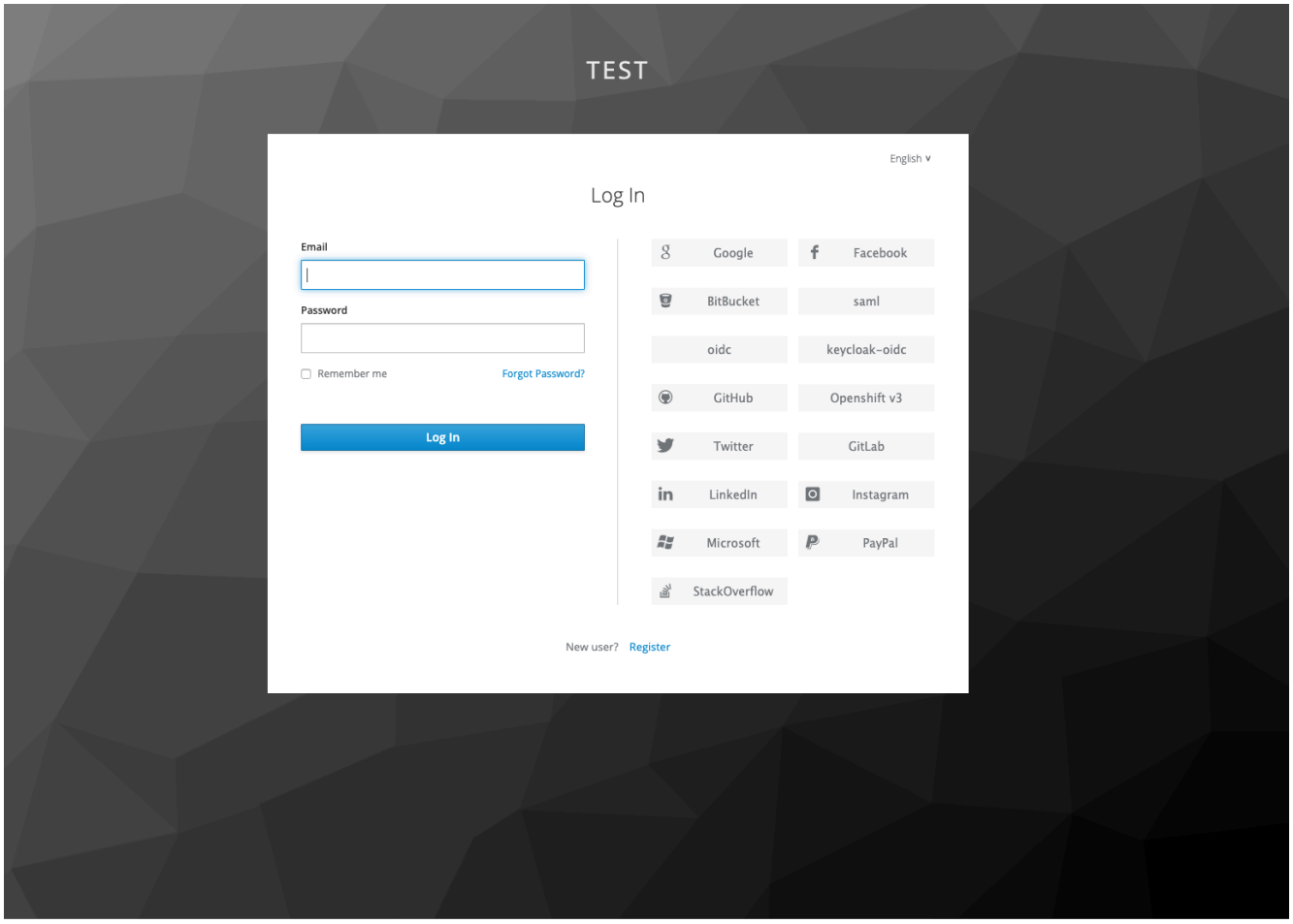
Yksi tapa käyttää Ansiblea on Playbookit. Niiden avulla voi asettaa erilaiset rakenteet, kertoa tehtävien suoritusjärjestyksen ja asettaa myös tehtävien uudelleensuorituksia tarpeen vaatiessa. Alla esimerkki Playbookista (KUVA 8). (Ansible 2019b.)

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service:
      name: httpd
      state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

KUVA 8. Esimerkki Ansiblen Playbookista. (Ansible 2019b)

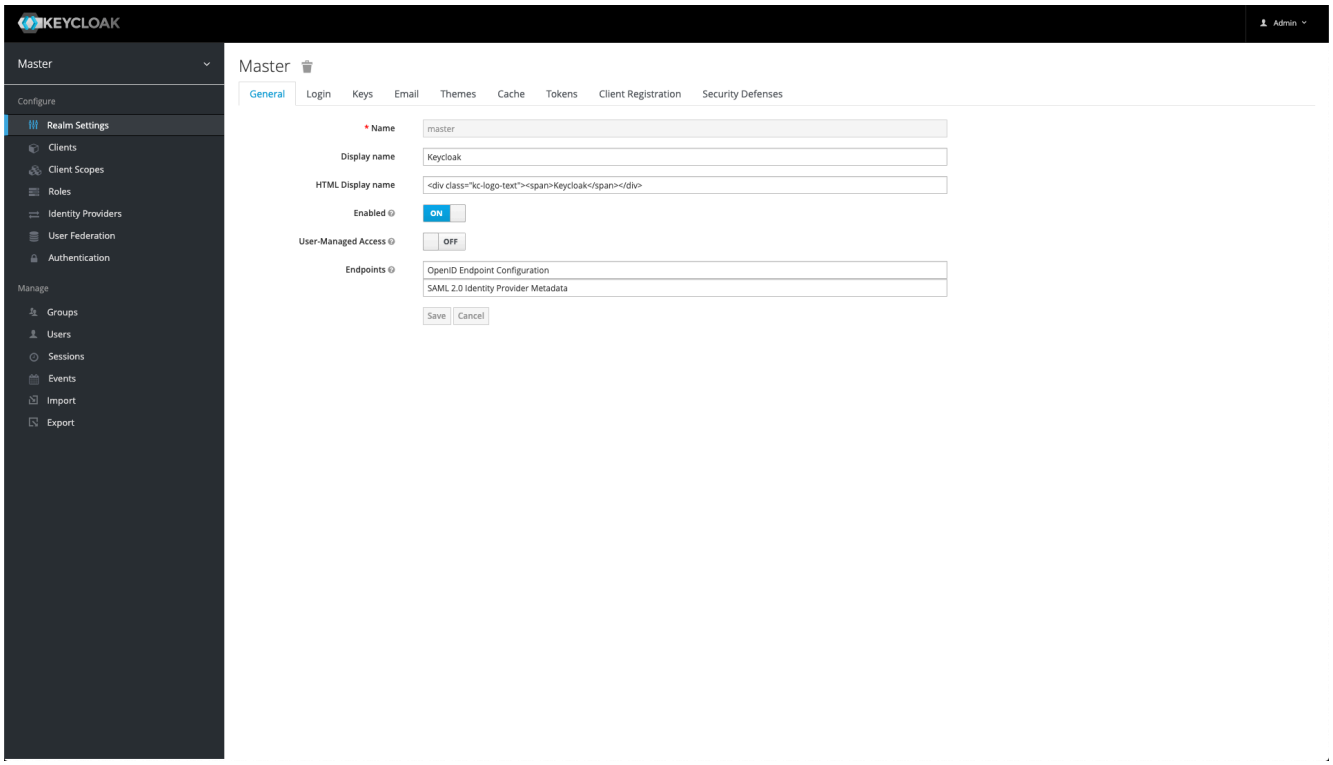
2.8 Keycloak

Keycloak on tunnistautumisjärjestelmä, jolla on avoin lähdekoodi. Keycloak tarjoaaa monia erilaisia ominaisuuksia näppärästi järjestelmänvalvojan konsolista valikoiden. Sen avulla voi luoda suljetumman kirjautumisen, johon voi kirjautua ainoastaan olemassa olevilla tunnuksilla. Siihen voi lisätä myös tuen rekisteröitymiseen luomalla tunnuksen ja salasanan. Sen lisäksi on myös mahdollista autentikoitua useiden eri sosiaalisten medioiden ja muiden tunnusten avulla. Niitä ovat muun muassa perinteiset Google, Facebook ja Twitter, mutta näiden lisäksi on esimerkiksi GitHub, GitLab, Paypal ja LinkedIn (KUVA 9). Käyttäjät on myös mahdollista tuoda LDAP:n tai Active Directorin serveriltä. Keycloak antaa tuen SAML:iin, OpenID Connectiin ja OAuth 2.0:aan. Se perustuu standardisoituun protokollaan. (Keycloak 2019.)



KUVA 9. Kirjautumisnäkymä, johon on lisätty kaikki mahdolliset kirjautumis- ja rekisteröitymisvaihtoehdot, salasanan palautus ja kielen vaihto.

Järjestelmänvalvojalla on käytössään konsoli (KUVA 10), josta hän pystyy hallitsemaan kaikkea, mitä Keycloakin sisällä tapahtuu. Sen lisäksi, että sieltä voidaan säätää kirjautumissivut ja autentikoitumis-vaihtoehdot, siellä on myös käyttäjien hallinta, sessioiden seuranta ja keskeyttäminen. Käyttäjiä voi myös lisätä ryhmiin ja heille voi antaa myös erilaisia rooleja ja sitä kautta antaa oikeuksia. (Keycloak 2019.)

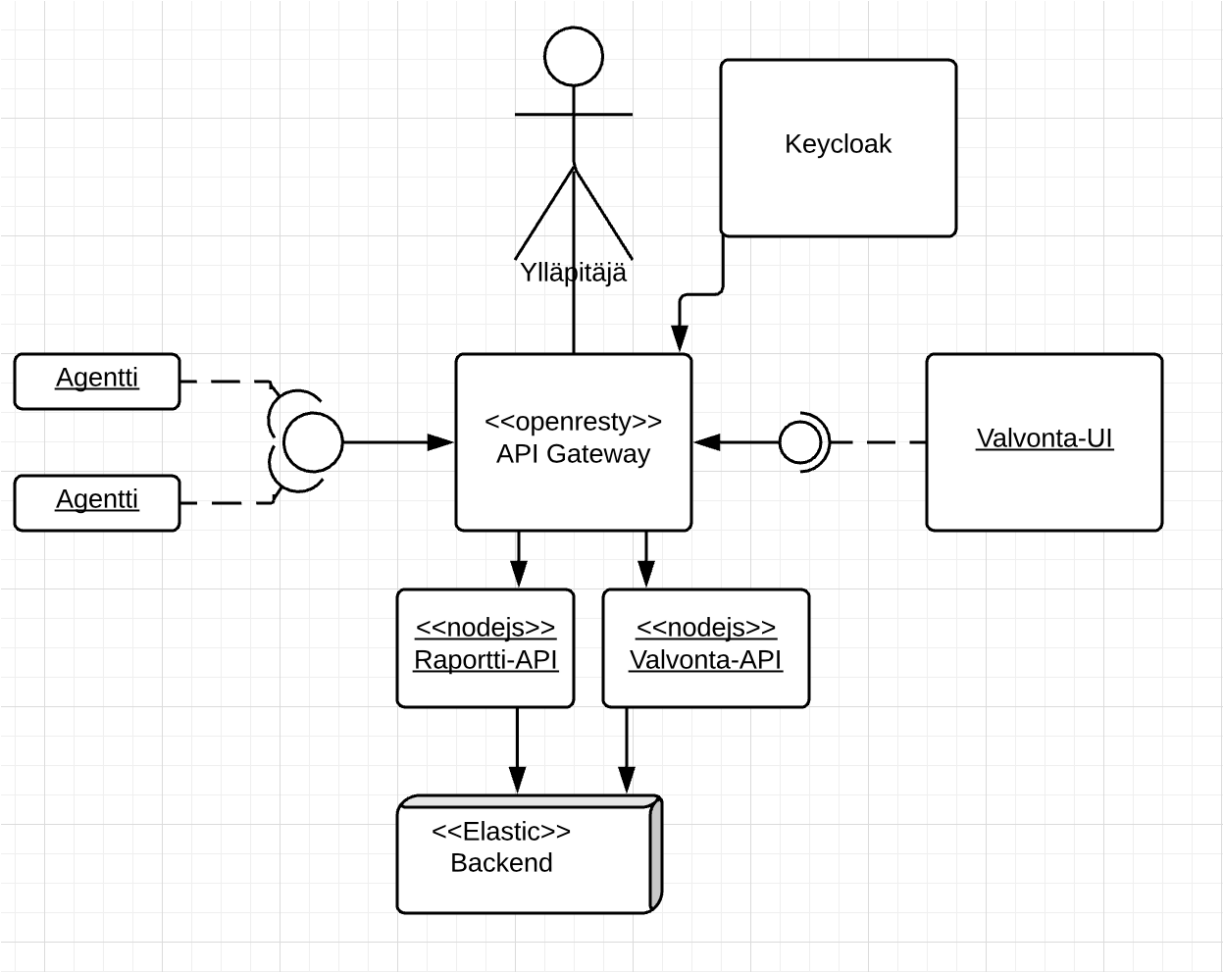


KUVA 10. Keycloakin järjestelmänvalvojan konsolin perusnäkymä.

3 KÄYTÄNNÖN TOTEUTUS

3.1 Suunnittelu

Opinnäytetyön tekeminen alkoi ajatustyöllä siitä, mitä toiminnallisuutta tavoitteet täyttävän sovelluksen tulee sisältää. Mitkä ovat ne todennäköiset käyttötapaukset, joiden tarpeisiin valittujen tekniikoiden pitäisi pystyä vastaamaan? Lopulta päädyin ratkaisuun, jossa dataa järjestelmään tuovat ja päivittävät automaatit ja käyttäjä voi olla niin Alfamen oma työntekijä kuin mahdollisesti myös asiakas. Tämän vuoksi yksi tärkeimmistä asioista on koko järjestelmän suojaus, koska ohjelmaa voidaan käyttää arkaluontoisten tietojen kanssa. Myös turhanpäiväinen datan lähetys häirintätarkoituksessa tietokantaan estyy, kun niin voivat tehdä vain ne, joilla on tunnus. Alla olevassa kuvassa esiintyvä agentti tulee olemaan myöhempää tuotantoa, kuten Valvonta-UI, mutta se lisättiin kuvaukseen tuomaan selkeyttä myös jatkokehitystä ajatellen (KUVA 11). Määrittelyni jälkeen kirjasin dokumentaation aiheesta.



KUVA 11. Alkuperäisen suunnitelman mukainen järjestelmä.

Valvonta- ja Raportti-API:n toiminnallisuuden luomisessa piti ottaa huomioon hyvä suorituskyky suurellakin kuormituksella. Lopulta pohdintojen jälkeen päädyin siihen lopputulokseen, että Raportti-API:n kevyin toteutus käytön kannalta olisi käytännössä uuden integraation luominen kantaan tietoineen ja olemassa olevan päivittäminen. Valvonta-API taas sisältää perustietojen haun, virhetiedon haun ja liikenteen haun.

3.2 Ohjelmointi

Tässä osassa käyn läpi ohjelmoinnin vaiheita ja ohjelmaan liittyviä eri osia ja niiden asetuksia. Tähän sisältyvät API:en ohjelmoinnit, gateway, backend ja autentikointi.

3.2.1 Valvonta-API

Aloitin toteuttamisen Valvonta-API:n luomisesta ja sen käyttötapauksien tarkistamisesta. Päädyin käyttämään Node.js:ää ja sen tukena express.js:ää, jotta saisin toteutettua API-kutsut eri skenaarioille. Ensin loin valvonta.yaml-tiedoston, jossa on kaikki API-kutsut esimerkkivastauksineen (KUVA 12). Sen kaveriksi rakensin server.js-tiedoston, jossa on linkitys tähän valvonta.yaml-tiedostoon. Nämä tiedostot yhdessä ohjaavat kutsuja oikealle toiminnallisuudelle. Tämän lisäksi loin jokaiselle Valvonta-API:n käyttötapaukselle oman express.js-tiedoston, jossa tapahtuu kaikki keskustelu Elasticsearchin kanssa ja josta lähtevät myös vastaukset takaisin käyttäjälle. Kyseisissä tiedostoissa on funktiot, joita server.js kutsuu yhteistyössä valvonta.yaml-tiedoston kanssa.

```yaml
info:
  description: API for database-to-UI connection
  version: 1.0.0
  title: Valvonta-API
host: 'localhost:3000'
basePath: /valvonta
tags:
  - name: integration
    description: Information about integration
schemes:
  - https
paths:
  /integrationsBasic:
    get:
      tags:
        - integration
      summary: Get information of integrations from database
      operationId: getIntegrations
      consumes:
        - application/json
      produces:
        - application/json
      parameters: []
      responses:
        '200':
          description: Returned basic information of integration
          schema:
            type: array
            items:
              $ref: '#/definitions/IntegrationBasic'
  /integrationsFailure:
    get:
      tags:
        - integration
      summary: Get information of failures from database
      operationId: getFailures
```

KUVA 12. Ote valvonta.yaml-tiedostosta.

Kutsun tullessa siis server.js vastaanottaa kutsun tiedot. Sitten se valikoi oikean reitin vertailemalla tietoja valvonta.yaml-tiedostosta löytyviin asetuksiin ja kutsuu funktiota saamiensa tietojen perusteella. Serverin kutsuttua funktiota funktio lähtee toimintaan, hakee vastauksen backendilta ja lähettää sen kutsun tekijälle (KUVA 13). Vastauksena palautuu JSON-muodossa olevaa tekstiä, jossa on kaikki kriteereihin sopineet tiedot. Esimerkin tapauksessa se hakee kaikki tiedot kaikista integraatioista.

```
module.exports = {
  get: function getIntegrations (req, res, next) {
    let body = {
        size: 10000,
        from: 0
      }
    let iBasic;

        elasticClient.search({index: 'integration', body:body, type:'_doc'})
        .then(results => {
        console.log(results);
          iBasic = results.hits.hits.reduce( ( acc, cur ) => {
            if( !acc[ 'integrations' ] ) acc[ 'integrations' ] = [];
            acc[ 'integrations' ].push( { id: cur._source.integration[0].id, name: cur._sourc
            return acc;
          }, {} );
            res.send(iBasic);

        })
        .catch(err=>{
          console.log(err)
          res.send([]);
        });
      }
};
```

KUVA 13. Ote integraatioiden tietojen hausta.

3.2.2 Raportti-API

Raportti-API on loppujen lopuksi hyvin samankaltainen toteutus kuin Valvonta-API:kin on. Siinä on myös raportti.yaml-tiedosto, jossa on API:en osoitteet, rakenteet ja vastaukset. Siellä on myös esimerkkirunko kutsusta, jollaisen se vastaanottaa. Myös kaikille käyttötapauksille on oma express.js-tiedosto, johon on koodattu niiden toiminta. Kutsun saapuessa server.js-tiedosto jälleen tutkii kutsun raportti.yaml-tiedoston avulla ja tarkistaa kutsun kohteen. Tässä tapauksessa se myös tarkistaa kutsun mukana tulevan datan rakenteen ja sisällön. Jos rakenne ja sisältö eivät täsmää esimerkin kanssa, käyttäjälle palautuu virhe, jonka perusteella se voi korjata omaa kutsuaan. Jos data on oikeassa rakenteessa ja sisältökin täsmää, se siirtyy raportti.yaml-tiedoston perusteella oikeaan express.js-tiedostoon ja sen sisältävään funktioon. Funktio tekee sitten datalle tarvittavat toimenpiteet (KUVA 14).

```
module.exports = {

    post: function postIntegration(req, res,) {

        const id = req.body.id

        const data = JSON.stringify( { 'integration': [ req.body ] } );

        elasticClient.index({
            index: 'integration',
            type: '_doc',
            body: data,
            id: id
            }).then(function(resp){
            return res.json(resp)
        }, function (err){
            console.log(err.message)
            return res.json(err.message)
        });
    }
};
```

KUVA 14. Ote integraation lähetyksestä kantaan.

3.2.3 Backend

Backendin päädyin toteuttamaan Elasticsearchilla. Tutkimisen ja vertailuiden jälkeen totesin sen toimivan näppärämmin tässä tarkoituksessa kuin esimerkiksi MongoDB. Node.js tukee hyvin Elasticsearchin liittämistä, ja sitä pystyy käyttämään hyvin monipuolisesti kantakyselyissä ja datan kantaan siirtämisessä. Se oli tärkein ominaisuus, mitä hain. Elasticsearch lähti käyntiin mukavasti ilman sen suurempia asetuksia. Käynnistys tapahtui Dockerin avulla. Dockerissa on Elasticsearchin kontti, jonka saa näppärästi käyntiin suorittamalla komennon ”docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it --name elk sebp/elk” komentokehotteessa. Sama komento löytyy myös Docker Hubin infosivuilta. Käynnistys ottaa jonkun aikaa, mutta käynnistyttyään Elasticsearch alkoi vastaanottaa dataa post-kutsuilta ja antaa dataa takaisin get-kutsuilla. Aiemmin nähdyissä kuvissa (KUVA 13 ja KUVA 14) nähdään, kuinka get- ja post-kutsuille on ohjelmoitu toiminta, joka kommunikoi Elasticsearchin kanssa. Elasticsearch saa tästä kutsusta tarvittavat tiedot, kuten indeksin, id:n ja varsinaisen datan. Niiden perusteella se tallentaa ja hakee tiedot kannastaan.

3.2.4 Gateway

Gateway toteutettiin tiukan aikataulun vuoksi kevyenä versiona, mutta siinä kuitenkin on tärkeimmät ominaisuudet. Se päästää läpi vain käyttäjiä, joilla on oikeus ja osaa ohjata oikeaan API:in ja siellä oikeaan osoitteeseen. Gateway on tällä hetkellä toteutettuna myös express.js:llä, kuten API:t. Mutta sen sijaan, että gateway varsinaisesti käsittelisi kutsut, se lähettää ne eteenpäin Express.js:stä löytyvällä instanssilla, joka on Router. Se on reitittävä väliohjelmisto. Lisäsin Gatewayhin lisäsin Keycloakin autentikoinnin, josta kerron lisää seuraavassa luvussa. Se onnistui helposti, sillä löysin npm:n avulla tähän tarkoitukseen sopivan komponentin. Asentamisen jälkeen lisäsin sen projektiin ja jokaiseen haluamaani kutsuun lisäsin autentikoinnin kirjoittamalla ”keycloak.protect()”. Sulut jätin tyhjäksi, sillä toistaiseksi haluan antaa kaikkien käyttäjien käyttää kaikkia kutsuja. Myöhemmin siihen todennäköisesti lisätään jokin ryhmätunniste, jotta vain annetun ryhmän jäsenet voivat käyttää kyseistä API:a.

Konfiguroin useita reittejä gatewayhin vastaamaan kaikkien API:en tarpeita, mutta tässä on esimerkki integraatioiden perustietojen hausta gatewayn kautta (KUVA 15). Ensimmäisellä rivillä näkyy osoite, jota kutsumalla laukaisee reitittimen käyntiin. Sen jälkeen on asetus keycloakista, jota sen täytyy käyttää. Onnistuneen autentikoinnin jälkeen se lähettää pyynnön halutulle API:lle ja vastauksen saatuaan lähettää sen eteenpäin käyttäjälle.

```
router.get('/valvonta/integrationsBasic', keycloak.protect(), function(req, res){
    api.get(req.path).then( resp => {
      res.send(resp.data)
    })
  })
```

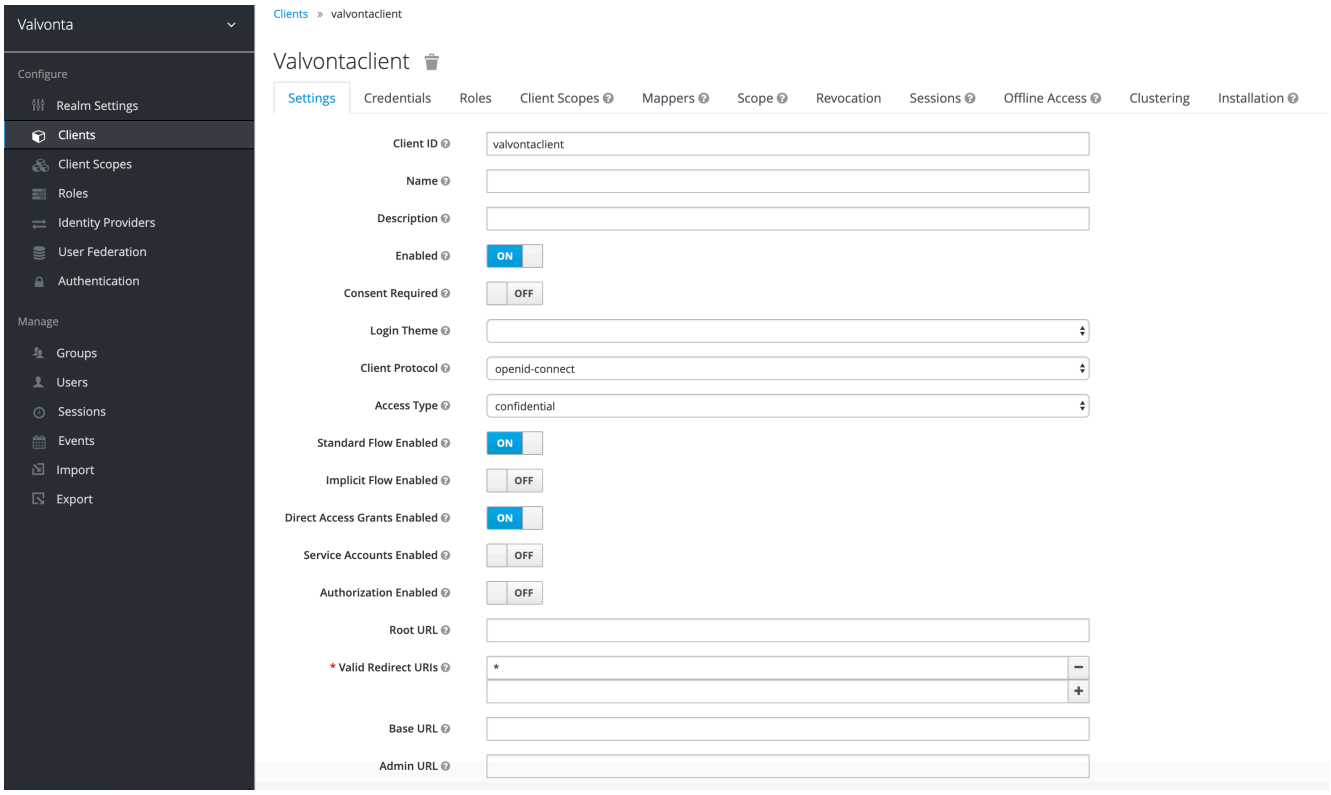
KUVA 15. Integraatioiden perustietojen haku gatewayn kautta.

3.2.5 Autentikointi

Autentikoinnin päätin toteuttaa eri vaihtoehtojen tutkimisen jälkeen Keycloakin avulla. Se tuntui olevan monipuolisin vaihtoehdoiltaan, ja se kytkeytyy hyvin express.js:n kanssa yhteen. Loin Keycloakin An-

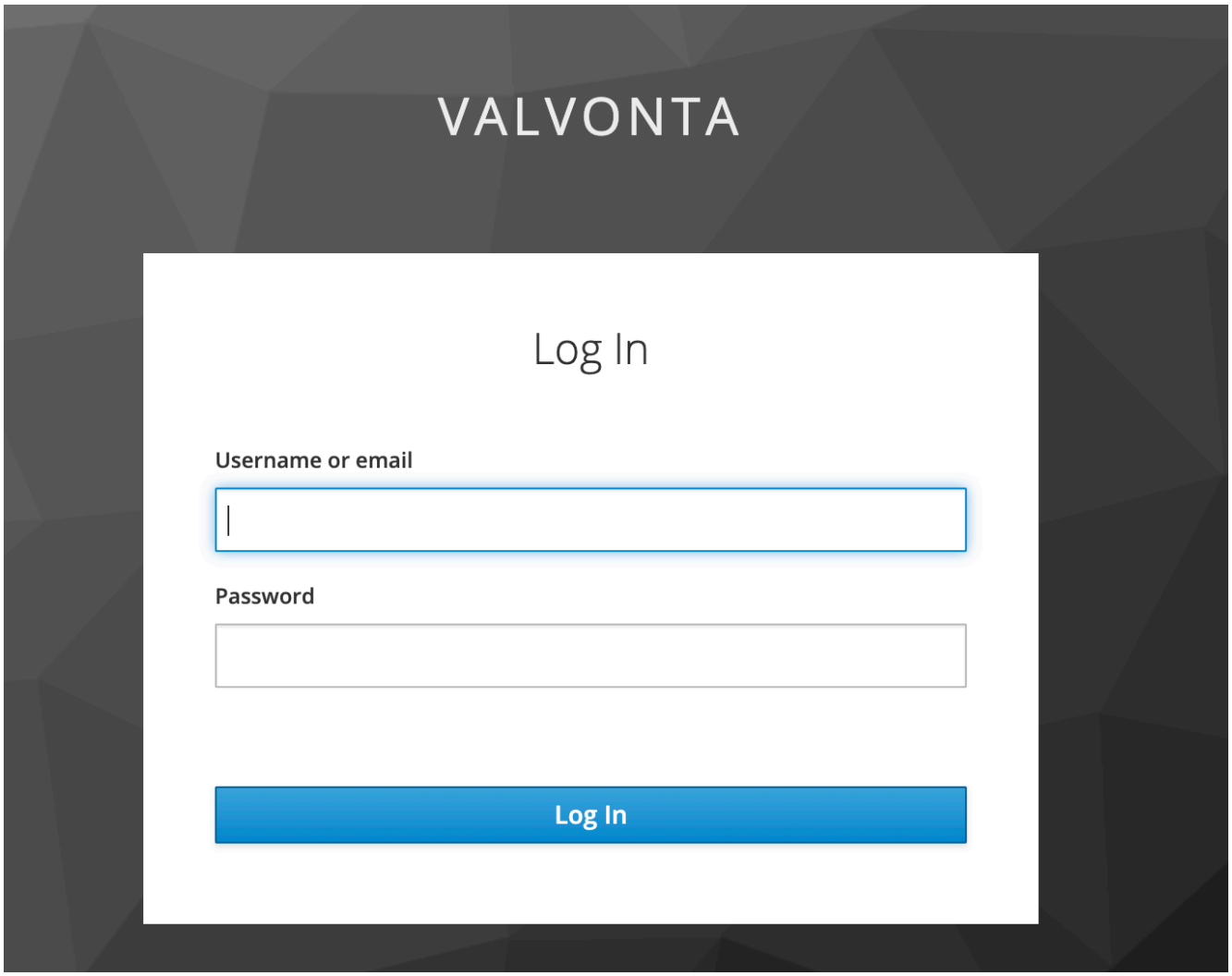
siblen Playbook-tiedostoja apuna käyttäen Alfamen sisäisessä kehityksessä käytössä olevaan virtuaalikoneeseen. Se vaati parin Playbook-tiedoston luomisen ja sen lisäksi pari asetustiedostoa, joiden perusteella Keycloak käynnistetään pystyyn ja otetaan käyttöön.

Tiedostojen luomisen jälkeen ensin ajettiin keycloakin asentava yml-tiedosto ansiblella. Ajon jälkeen asetuksissa osoitetusta osoitteesta löytyy Keycloakin järjestelmänvalvojan konsoli. Sinne loin uuden hallinta-alue ja nimesin sen Valvonnaksi projektin mukaisesti. Valvonnan alle loin uuden clientin, joka tulee vastaamaan Gatewayn tarpeisiin. Asetuksia clientissä on todella paljon, joten tässä kohdassa joutui turvautumaan tarkemmin Keycloakin dokumentaatioon tarvittavien asetusten löytämiseksi (KUVA 16). Ensimmäiseltä sivulta täytyi asettaa pääsytyyppi luottamukselliseksi ja uudelleenohjausosoitteen asetin arvoon *. Tällä arvolla Keycloak hyväksyy kaikki uudelleenohjausosoitteet. Turvallisuuden kannalta tällainen menettely ei ole hyvä. Tämän tyyppisessä kevyessä ratkaisun käyttökelpoisuuden todistamisessa se on hyväksyttävää niin kauan kuin ohjelmaa ajetaan vain paikallisesti omalla tietokoneella vailla yhteyttä ulkomaailmaan.



KUVA 16. Näkymä Valvontaclientin asetusten perusnäkymästä.

Tallennettuani yllä mainitut asetukset yläpalkkiin ilmestyy välilehti ”credentials”. Se näkyy jo aiemmassa kuvassa (KUVA 16), sillä olin jo tallentanut asetukset. Siellä on vaihtoehdot autentikoitumiseen. Käytin tässä projektissa yksinkertaisesti tietoja client ID ja secret. Credentials-välilehdeltä saa kopioitua secret-merkkijonon, jonka asetin seuraavan ansible-ajon asetustiedostoihin. Ajon jälkeen kirjautuminen toimii luomillani käyttäjätunnuksilla. Nyt kun käynnistää gatewayn komentokehotteesta, hakee integraation perustietoja sen kautta, avautuu Keycloakin näkymä, jossa pystyy kirjautumaan sisään (KUVA 17).



KUVA 17. Näkymä haettaessa gatewayn takaa tietoja.

Todennuksen toteuttamisen kanssa tuli jonkun verran haasteita konsolin asetustem ja itse allekirjoitetun varmenteen vioksi. Asetuksia on todella suuri määrä konsolista ja ohjeita on myös paljon. Vei jonkin verran aikaa löytää juuri ne itseä kiinnostavat asetukset, mutta onneksi Keycloakilla on hyvä dokumentaatio. Myös keycloakista ja sen eri käyttötavoista tuntuu olevan ympäri internettiä paljon keskusteluita,

joista löytyi apuja. Saatuani ensimmäisen kerran todennuksen käyttöön törmäsin ongelmaan varmenteen kanssa. Pitkän tutkimisen jälkeen pääsin lopulta jyvälle siitä, miten voin kirjoittaa uuden varmenteen, jonka Keycloak hyväksyy. Siihen löytyi useita ohjeita, mutta tapa valikoitui express.js:n ja Node.js:n kautta. Lopulta kutsu kulki ongelmitta läpi ja vastauksena palautuivat integraatioiden perustiedot, joita oltiin hakemassakin.

3.3 Testaus ja jatkokehityksen tukitoimet

Testausta suoritin koko projektin ajan pienemmissä kokonaisuuksissa. API:t testasin erikseen pystytettyäni Elasticsearchin. Ensin testasin Raportti-API:n ja Elasticsearchin yhteistoiminnan lähettämällä dataa Raportti-API:n avulla. Vastauksen perusteella yhteys näytti toimivan mallikkaasti. Vahvistuksen asialle sain samalla, kun testasin Valvonta-API:n ja Elasticsearchin yhteyttä. Vastauksena sain dataa, jonka olin aiemmin lähettänyt Elasticsearchiin Raportti-API:n avulla (KUVA 18). Näiden valmistumisen ja testaamisen jälkeen toteutin gatewayn ja sen todennuksen. Näiden toteuttamisen jälkeen testasin koko reitin Raportti-API:sta Valvonta-API:in gatewayn kautta todennettuna.


```
{
  - integrations: [
      - {
            id: "id2",
            name: "name",
            organisation: "organisation",
            direction: "inbound",
            setFrequency: 1,
            timestamp: "2000-01-23T04:56:07.000Z"
        },
      - {
            id: "id",
            name: "name",
            organisation: "organisation",
            direction: "inbound",
            setFrequency: 1,
            timestamp: "2000-01-23T04:56:07.000Z"
        }
    ]
}
```


KUVA 18. Vastaus selaimelta haettaessa integraatioiden perustietoja.

Todettuani koko ketjun toimivaksi alusta loppuun tein yksityiskohtaisen ohjeistuksen nykyisen toteutuksen käyttöön ottamisesta. Näin kuka tahansa Alfamella voi ottaa projektin jatkokehitettäväksi. Lisäksi päivitin tikettijärjestelmän seuraavista askelista ja liitin määritysdokumentin dokumentaatioon.

4 YHTEENVETO

Tämä opinnäytetyöprojekti oli todella mielenkiintoinen toteuttaa, ja sen aikana pääsi testaamaan ja käyttämään useita mielenkiintoisia tekniikoita. Osa tekniikoista oli jo entuudestaan jonkin verran tuttuja työkokemuksen myötä, mutta niiden tuntemus ja käyttö tulivat huomattavasti tutummiksi opinnäytteen myötä. Osa tekniikoista oli täysin uusia tuttavuuksia ja vaati paljonkin tutkimustyötä ennen kuin niitä pääsi kunnolla käyttämään. Varsinkin Keycloak oli mielenkiintoinen autentikointiratkaisu, jota todennäköisesti tulen käyttämään myös jatkossa erilaisissa projekteissa autentikoinnin toteuttamiseen. Alun haasteiden jälkeen käyttö on jo huomattavasti sujuvampaa ja laajat käyttömahdollisuudet houkuttelevat.

Projektista alun perin tehty määrittelydokumentti poikkeaa lopullisesta tuotoksesta useammastakin syystä. Syihin lukeutuu muun muassa se, että joihinkin asioihin meni odotettua kauemmin. Sen lisäksi oli myös tiedossa, että UI:n tekemiseen aika tuskin riittäisi, mutta päätin sen myös kirjata määrittelydokumenttiin jatkokehitystä varten. Myös yllättäen muuttuvat tilanteet yrityksessä muokkasivat jonkin verran toteutusta, kun rinnalle suunniteltu opinnäytetyö jouduttiin yllättäen perumaan. Näistä muutoksista johtuneesta stressistä huolimatta selvisin ja jatkoin työskentelyä. Tämä kokemus tukee varmasti myös jatkossa tulevissa projekteissa, joissa varmasti tulee tapahtumaan muutoksia täysin yllättäen.

Kokonaisuudessaan tämä opinnäytetyöprojekti oli erittäin opettavainen. Siinä pääsi hoitamaan monenlaista roolia, jotka esiintyvät projektin eri vaiheissa. Arkkitehdin hommia pääsi koittamaan määrittelyssä, kun piti tutkia, mitä tekniikoita on järkevä käyttää ja toimivatko ne yhteen. Siihen kuului myös määritysdokumentin työstäminen. Sen lisäksi sitten oli tietenkin ohjelmoijan rooli varsinaisena toteutuksena ja asiakkaalle raportoiminen, joka tässä tapauksessa on Alfame Systems Oy. Projektiin jäi vielä monenlaista toteutettavaksi ja selkeitä kohteita, joita voisi parannella. Näitä asioita tulen varmasti jatkossa vielä kehittämään ja viemään eteenpäin.

LÄHTEET

Ansible. 2019a. Why Ansible. Saatavilla: https://www.ansible.com/overview/it-automation. Viitattu 25.11.2019.

Ansible. 2019b. Intro to Playbooks. Saatavilla: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html. Viitattu 25.11.2019.

Docker. 2019a. Docker: The modern platformfor high-velocity innovation. Saatavilla: https://www.docker.com/why-docker. Viitattu 25.11.2019.

Docker. 2019b. Build and Ship any Application Anywhere. Saatavilla: https://hub.docker.com. Viitattu 25.11.2019.

Elastic. 2019a. What is Elasticsearch? Saatavilla:https://www.elastic.co/what-is/elasticsearch. Viitattu 25.11.2019.

Elastic. 2019b. Elasticsearch Service pricing. Saatavilla: https://www.elastic.co/products/elasticsearch/service/pricing. Viitattu 25.11.2019.

Keycloak. 2019. About. Saatavilla: https://www.keycloak.org/about.html. Viitattu 25.11.2019.

MDN. 2019. Express/Node introduction. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. Viitattu 25.11.2019.

Nodejs. 2019a. Brief history of nodejs. Saatavissa: https://nodejs.dev/a-brief-history-of-nodejs. Viitattu 25.11.2019.

Nodejs. 2019b. Releases. Saatavissa: https://nodejs.org/en/about/releases/. Viitattu 25.11.2019.

Npm. 2019a. Npm products. Saatavissa: https://www.npmjs.com/products. Viitattu 25.11.2019.

Npm. 2019b. About npm. Saatavissa: https://docs.npmjs.com/about-npm/. Viitattu 25.11.2019.

VirtualBox. 2019. Chapter 1. First steps. Saatavilla: https://www.virtualbox.org/manual/ch01.html. Viitattu 25.11.2019.